Journal of Computational and Applied Mathematics 26 (1989) 311–325
North-Holland

Linear algebra software on IBM and CRAY computers

J.-Fr. HAKE and W. HOMBERG
Zentralinstitut für Angewandte Mathematik (ZAM), Kernforschungsanlage Jülich GmbH (KFA), D-5170 Jülich, Fed. Rep. Germany

Received 27 July 1988
Revised 3 February 1989

Abstract: Mathematical software libraries represent collections of subprograms for the solution of frequently occurring numerical problems. The acceptance of mathematical software libraries strongly depends on the quality of their components, because the library routines are often used as building blocks in more complex computer codes. This approach can simplify code development and introduce know-how to the user's code. Results on the performance and accuracy of library subroutines from IMSL and NAG for matrix multiplication, solution of linear equations, and eigenvalue problems on a CRAY X-MP and an IBM 3081 are reported.

Keywords: Mathematical software, numerical methods, linear algebra, matrix multiplication, linear equations, eigenvalue problems, vector computer.

1. Introduction

Mathematical software libraries (IMSL, NAG) supply subprograms for the solution of a broad class of frequently occurring numerical problems. Therefore, the library routines are normally used as building blocks in more complex computer codes. Users taking advantage of this approach can simplify code development while being assured that the quality of code is high. The libraries can be grouped into three categories: machine-dependent libraries provided by the manufacturer of a computer system (SCILIB), installation-dependent libraries developed and maintained by the staff of a local computer center (ZAMLIB) and software-house libraries (IMSL, NAG) covering a broad spectrum of routines on a variety of computer systems for mathematical and statistical problems [1,2,8,11].

The acceptance of mathematical software libraries is strongly related to the quality of their components with respect to efficiency, accuracy, portability, reliability, and robustness.

Efficiency of a library subroutine can be judged from

(1) computational complexity and storage requirements of the selected algorithm, i.e., for numerical problems the size of computer memory and the number of floating point operations required to compute a solution;

(2) the implementation of the algorithm on a specific computer system. CPU-time consumption (MFLOP-rate) and storage efficiency (bank conflicts, locality) represent indicators for well-implemented algorithms.

Accuracy considerations have to accompany the computational process. Model problems with analytical solutions represent a tool to estimate the error introduced during the computation. For linear algebra problems, there exists a large collection of matrices which can be used for various tests in this field [9]. The following error criterion is convenient: for vectors $a = (\alpha_i) \in \mathbb{R}^n$, a norm can be defined by $\| a \| := \max_i | \alpha_i |$. The absolute error ϵ of an approximation b to a is then given by $\epsilon := \| a - b \|$.

Portability remains a major goal for the design of library routines. But in many cases, software squeezing the most out of a high-performance computer has to reflect the underlying architecture. For the user-interface, this conflict can at least partially be resolved by introducing standardized naming conventions and parameter lists. On the algorithmic level, the BLAS concept represents an approach to introduce efficiently programmed basic mathematical operations into complex algorithms [3,14].

Reliability and robustness of library software measure how well the software performs its intended functions resp. the degradation in performance as usage moves away from the intended domain of use. Both properties are not of interest to this report.

This report is concerned with the performance of linear algebra software from IMSL and NAG. Both libraries have been implemented on CRAY and IBM computer systems in order to analyze the differences between the two libraries on a classical scalar and a modern vector computer architecture, respectively. The subsequently described results have been obtained from tests with software from IMSL Edition 10 [1] [13], NAG Mark 12 [1] [16], and some other programs. All tests have been run on a CRAY X-MP/22 (operating system COS 1.16 BF1, FORTRAN compilers CFT 1.15 and CFT77 Version 1.3) and an IBM 3081 (operating system MVS, VS FORTRAN compiler Version 1.4.1 Optimizing Level 2) at ZAM of KFA Jülich. A comparison of results is based on the IBM double precision and CRAY single precision vendor-supplied [2] versions of the libraries.

Each of the two CRAY processors has a cycle time of 9.5 nsec resulting in a theoretical peak performance of 210 MFLOPS per processor which in practice approximately can be achieved, if vector instructions are used extensively. The central memory in the CRAY computer (2 MW of storage) is organized in 16 banks. Memory access conflicts (bank conflicts) can significantly slow down the performance of user-written programs. For some programs, the sensitivity to bank conflicts depends on the actual size of the allocated arrays. This effect has been studied by running all test problems on one processor with exact dimensioning of the required FORTRAN arrays.

The IBM 3081 model D has two processors; each of them has a cycle time of 26 nsec. On this computer, only operations with scalar operands can be executed. The size of the main memory is 32 MB. High-speed data transfer between registers and main memory is supported by a cache of 32 KB. Again, all test problems have been solved on one processor with exact dimensioning of the FORTRAN arrays. Besides the differences in the design of the processors, the two different conceptions of memory organization (interleaved memory on CRAY and hierarchical memory on IBM) may also affect the performance of programs. These effects are reported, if they have been observed.

[1] Implemented on high-speed computing facilities at KFA Jülich/ZAM.
[2] Implemented as recommended in the corresponding installation guide.

On many computer systems, the measured CPU-times are workload dependent and vary within a certain range. For this report, timings obtained from jobs running in batch mode are measured with routine SECOND. On CRAY, SECOND is a member of the SCILIB [2], the IBM version of SECOND is a KFA specific time measuring routine supplying analogous information. CRAY MFLOP-rates are derived from this data counting additions and multiplications.

2. Matrix multiplication

Matrix multiplication may be used as a starting point for more complex studies. The product $C = (\gamma_{ij}) \in \mathbb{R}^{n \times m}$ of two rectangular matrices $A = (\alpha_{ik}) \in \mathbb{R}^{n \times p}$ and $B = (\beta_{kj}) \in \mathbb{R}^{p \times m}$ is defined by the relation $\gamma_{ij} = \sum_{k=1}^{p} \alpha_{ik} \beta_{kj}$. The simply structured algorithm can be implemented in six different ways depending on the order of the FORTRAN DO-loops for the indices i, j, k. Different versions of the algorithm are denoted by the corresponding triple, i.e., *jki*-version means that the outer loop is referenced by index j and the innermost loop by index i [5].

- F01CKF (NAG) represents a *jki*-version of the algorithm, where the innermost loop is a SAXPY operation. On the CRAY, the vector loop for index k has been unrolled to a depth of four [16].
- (D)MRRRR (IMSL) also represents a *jki*-version. On the CRAY, unrolling for index k to a depth of 16 has been implemented [13].
- MM (Dongarra) is based on the same mechanism of column-oriented access to the matrices. MM uses a subroutine SMXPY which is a simplified form of Level 2 BLAS SGEMV. Unrolling is not used [4].
- VMULFF (predecessor of MRRRR). The *ijk*-variant is implemented in subroutine VMULFF, where the innermost loop represents an SDOT-operation (index k).

On the CRAY, F01CKF and MM show similar performance. IMSL and NAG use a Level 2 BLAS routine SGEMV (matrix-vector product) for the two innermost loops. Unrolling reduces CPU-time consumption of MRRRR by 25% compared with subroutine F01CKF. Within NAG's version, a SAXPY operation is only executed for nonzero scalars. For sparse problems, this more elaborate control structure may lead to reduced CPU-time consumption. To demonstrate the impact of this slight modification, CPU-times have also been measured for a test problem where the right operand represents a tridiagonal matrix; i.e., for matrix multiplication, the number of floating-point operations is now of order $O(n^2)$. Timings have been added to Table 1 in parentheses.

On IBM, F01CKF and DMRRRR consume about the same amount of CPU-time. For large n, the slightly worse performance of DMRRRR is due to a less efficient implementation of the conditional DAXPY operation which now is also supplied by the IMSL. Subroutine MM does not use temporary variables for loop independent array elements resulting in 10% overhead.

More fundamental differences can be observed from a comparison of the *ijk*-and *jki*-versions. Table 2 provides CPU-times measured on both computer systems for these two variants and $206 \leqslant n \leqslant 211$. On both computers, the AXPY-based programs behave in a very smooth way. For the DOT-based subroutine VMULFF, a higher CPU-time consumption has been measured which mainly consists of two components:

Table 1
CPU-time (sec) for matrix multiplication (quadratic case) on IBM 3081 and CRAY X-MP/22

IBM				
N	F01CKF (NAG Mark 12)	DMRRRR (IMSL Ed. 10)	MM (Dongarra)	VMULFF
150	3.04	3.19	3.46	3.85
200	7.20	7.43	8.18	9.60
250	14.04	14.36	15.99	21.52
300	24.18	24.83	27.74	48.59
350	38.35	39.45	43.91	82.21
400	56.95	57.78	65.49	–
CRAY				
N	F01CKF (NAG Mark 12)	MRRRR (IMSL Ed. 10)	MM (Dongarra)	VMULFF
250	0.25 (0.04)	0.19	0.27	0.57
300	0.44 (0.05)	0.32	0.46	0.92
350	0.68 (0.06)	0.50	0.68	1.38
400	1.00 (0.08)	0.74	1.01	3.67
500	1.84 (0.13)	1.40	1.92	5.75
600	3.15 (0.18)	2.40	3.31	7.70

(1) the generated code is more complex leading to about 20% increased CPU-times on the IBM; for the CRAY, differences in code generated for SAXPY and SDOT by the CFT compiler have been described earlier [12];

(2) besides the differences in the complexity of the generated code, memory access conflicts lead to a more irregular behaviour in CPU-time consumption. On a CRAY, peaks in CPU-time ($n = 208 = 16 \times 13$) occur if n is a multiple of the number of banks (16 on a CRAY X-MP/22 and 32 on a CRAY X-MP/48) and the matrix A is referenced by rows. The sensitivity to bank conflicts is indicated by the fact that for some increasing values of n the consumed CPU-time decreases significantly. On the IBM 3081, peaks occur more irregularly because of the LRU cache replacement strategy. For $n = 210$, the number of cache misses is about 8 times as large as for the succeeding value of n.

Table 2
CPU-time (sec) for matrix multiplication based on AXPY and DOT kernels

N	IBM 3081		CRAY X-MP/22	
	DAXPY kernel	DDOT kernel	SAXPY kernel	SDOT kernel
206	8.63	10.68	0.16	0.28
207	8.73	10.54	0.16	0.25
208	8.86	11.38	0.16	0.50
209	9.50	14.11	0.16	0.23
210	9.57	17.98	0.16	0.27
211	9.33	12.01	0.19	0.29

3. Linear systems with a symmetric positive definite matrix

The direct solution of a system of linear equations with a symmetric positive definite matrix A is based on the Cholesky decomposition and can be achieved by the following subroutines:

- (D)L2LDS (IMSL). First A is decomposed using (D)L2CDS. The solution is computed by forward/backward substitution ((D)LFSDS). In addition, an estimated condition number is computed [7].
- F03AEF, F04AGF (NAG). Based on the Cholesky decomposition (F03AEF), the solution is computed by forward/backward substitution (F04AGF). A determinant is computed. On a CRAY, both subroutines use unrolled vector loops with depth four [18, pp. 31–44].

Moreover, two high-accuracy solvers are considered:

- (D)L2ADS (IMSL). Starting with a Cholesky decomposition ((D)LFCDS), the system of equations is solved by forward/backward substitution ((D)LFSDS). This solution is improved by iterative refinement ((D)LFIDS). The iteration is terminated after 50 steps or if the relative change, measured in the maximum norm, in the computed solution is less than $\epsilon = 0.7 \times 10^{-14}$ on CRAY or $\epsilon = 0.2 \times 10^{-15}$ on IBM. For the iterative refinement, dot products are accumulated in extended precision hence disallowing vectorization on the CRAY. An estimated condition number is computed.
- F04ABF (NAG). Starting with a Cholesky decomposition (F03AEF), the system of equations is solved by forward/backward substitution (F04AGF). This solution is improved by iterative refinement. The iteration is terminated in case of nonconvergence detected by insufficient small corrections (the new relative change, measured in the maximum norm, may not exceed one half of the previous one) or if the maximum relative change is less than 2ϵ. Residuals are available to the user. For the iterative refinement, dot products are accumulated in extended precision hence disallowing vectorization on a CRAY [18, pp. 31–44].

On CRAY, L2LDS consumes twice as much CPU-time as the corresponding NAG combination F03AEF, F04AGF to be understood as caused by differing implementations of the Cholesky factorization. In the NAG library, the lower triangular Cholesky factor is generated columnwise; the procedure is based on matrix-vector multiplications using Level 2 BLAS SGEMV (unrolling to depth four). IMSL computes the upper triangular factor columnwise by solving a sequence of linear systems with triangular coefficient matrices; Level 2 BLAS STRSV (no unrolling) is called. The two Level 2 BLAS routines differ in their Level 1 kernels; SGEMV is based on a SAXPY operation whereas in this context STRSV uses a dot product. A comparison of code generated by the CRAY CFT compiler for BLAS operations using a SDOT or a SAXPY operation for their innermost loop has already been mentioned. For $n = 601$, F03AEF, F04AGF operates at a rate of 113 MFLOPS while the routine L2LDS achieves a speed of 76 MFLOPS.

On the IBM computer, timings for F03AEF, F04AGF and DL2LDS do not differ significantly. In analogy to the CRAY versions of the libraries, Level 2 BLAS is used. For large n, DL2LDS becomes slightly superior because of its very fast Level 1 BLAS kernel DDOT. The dot product is computed by an unrolled loop (depth five) leading to a very efficient IBM-version of DL2LDS. Supplementary tests based on DDOT without unrolling, result in worse performance than observed for F03AEF, F04AGF.

From Fig. 1 one may conclude that on a CRAY vector computer, iterative refinement becomes the dominating part when computing a high-accuracy solution. This observation can be analyzed

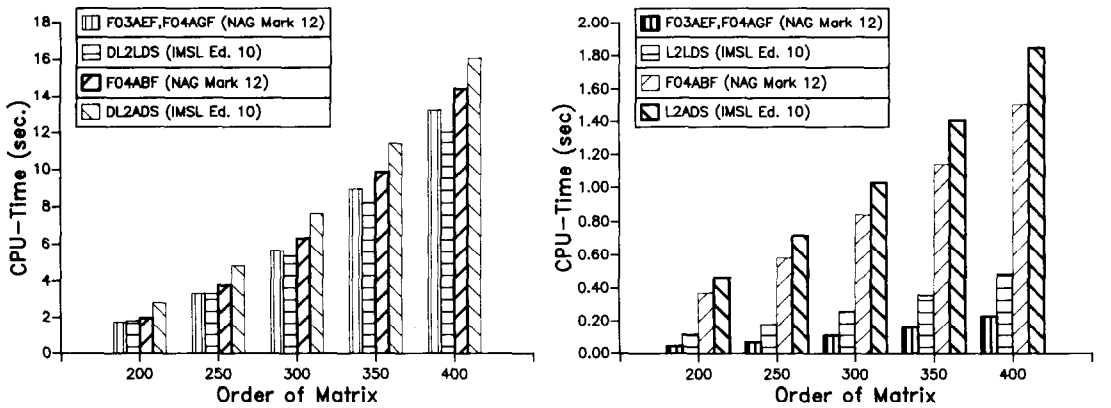

Fig. 1. CPU-time for the solution of linear systems (symmetric positive definite case) on IBM 3081 and CRAY X-MP/22.

with help of Table 3 showing the resource consumption of iterative refinement (number of iterations multiplied by CPU-time per iteration). Computation of residuals in extended precision represents the time consuming step of iterative refinement. Its key operation is a sequence of dot products which are accumulated in extended precision, X03AAF for NAG and SDDOTI, SDDOTA for IMSL. IMSL's CRAY single precision version uses a quadruple precision accumulator, i.e., a double precision array of length two, where a single precision array of length two (X03AAF) seems to be sufficient. Replacing SDDOTI, SDDOTA by X03AAF, the IMSL routine would probably turn out to be superior to be NAG routine for sufficient large n. On IBM, IMSL uses the same expensive approach; here, routines DQDOTI and DQDOTA are called.

Differences between CRAY and IBM in treating extended precision can be characterized by the number of single (double) precision floating-point operations which can be executed in the same time as one extended floating-point operation. For an IBM 3081, this ratio is about 1.9 while for a CRAY X-MP/22, the value of the quotient is about 19 leading to the dominance of iterative refinement ($O(n^2)$) over the direct solution ($O(n^3)$) of linear systems. Up to $n = 600$, one iteration is more expensive than the solution of a linear system of same order.

For $n = 601$, F04ABF operates at a rate of 23 MFLOPS (3 iterations), while the routine L2ADS achieves a speed of 22 MFLOPS (2 iterations).

The relative error in the solution computed by F03AEF, F04AGF and (D)L2LDS is 10^{-11} on IBM and 10^{-10} on CRAY.

Table 3
CPU-time consumption of iterative refinement (symmetric positive definite case)

N	IBM 3081		CRAY X-MP/22	
	F04ABF	DL2ADS	F04ABF	L2ADS
300	3*0.26 sec.	2*1.13 sec.	3*0.24 sec.	2*0.38 sec.
350	3*0.36 sec.	2*1.56 sec.	3*0.33 sec.	2*0.53 sec.
400	3*0.48 sec.	2*2.05 sec.	3*0.43 sec.	2*0.70 sec.

4. Linear systems with a real square matrix

For all libraries under consideration, the solution of a system of linear equations with a regular coefficient matrix is based on variants of the LU-decomposition and can be achieved by the following subroutines:

- (D)L2LRG (IMSL). This subroutine is based on LU-factorization with scaled partial pivoting (L2CRG). The resulting systems are solved by forward/backward substitution (LFSRG). An estimate for the condition number is computed [7,15].
- F04AAF (NAG). The algorithm is based on Crout's factorization (L is lower triangular, U is unit upper triangular) with row scaling and partial pivoting. Then, the solution is found by forward/backward substitution. Extended precision accumulation of inner products is not used. For the CRAY, vector loops are unrolled to a depth of four [18, pp. 93–10].
- LU, LUS (Dongarra). Crout's factorization with partial pivoting; the system is solved by forward/backward substitution (no unrolling).

In addition, two high-accuracy solvers are considered:

- (D)L2ARG (IMSL). First, the LU-factorization with scaled partial pivoting is computed (L2CRG). The resulting systems are solved by forward/backward substitution (LFSRG). LFIRG invokes iterative improvement, where inner products are accumulated in extended precision inhibiting vectorization on a CRAY. The iteration is terminated if the relative change in the computed solution is less than $\epsilon = 2^{-47}$ for all components; the maximum number of iterations is set to 50. An estimate for the condition number is computed.
- F04AEF (NAG). Iterative refinement is used additionally to improve the accuracy of the solution. Inner products are accumulated in extended precision inhibiting vectorization on a CRAY. The iterations are halted if they are found to be nonconvergent or convergent according to the criterion mentioned for subroutine F04ABF [18, pp. 93–110].

From Fig. 2 one may conclude, that on CRAY, L2LRG and F04AAF operate approximately at the same speed. For large n, L2LRG becomes slightly superior because the factorization step is more efficient. Here, the upper triangular matrix of the factorization is generated by a sequence of rank-one updates; Level 2 BLAS SGER (no unrolling) is called. In contrast to IMSL, NAG computes both factors of the LU-decomposition columnwise using Level 2 BLAS

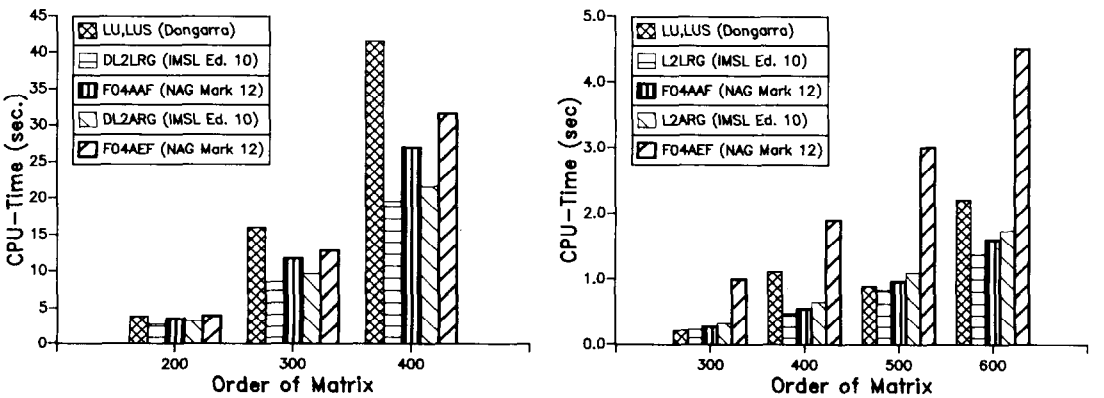

Fig. 2. CPU-time for the solution of linear systems (real quadratic case) on IBM 3081 and CRAY X-MP/22.

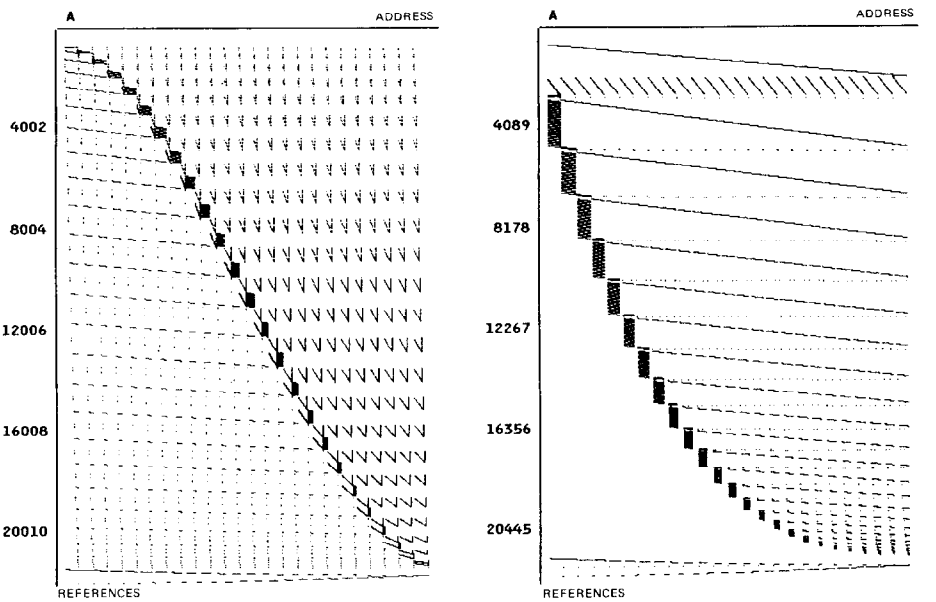

Fig. 3. Storage access patterns for LU, LUS and L2LRG.

STRSV (unrolled to depth four) for the strict upper triangle and SGEMV for the lower triangle. In contrast to the symmetric positive definite case, the Level 1 BLAS kernel of STRSV now represents a SAXPY operation. Storage references of the subroutine L2LRG are given by the right part of Fig. 3 and are contrasted with the storage access pattern of LU, LUS. The horizontal axis represents the storage locations for the coefficient matrix A (columnwise); the vertical axis gives the number of references.

The algorithm represented by LU, LUS may be decomposed into three steps:

(1) searching for pivot indices, interchanging rows and updating the pivot columns in the lower triangle indicated by slowly decreasing broken lines and blocks of short lines on the diagonal of the diagram;

(2) updating the pivot rows in the upper triangle of the matrix marked by sequences of 'V's;

(3) forward/backward substitution marked by two dashed lines at the lower end of the figure.

Bank conflicts arise from that part of the Crout decomposition where the upper triangular factor is generated (item (2)). They occur from rowwise access to the matrix and can be observed in Fig. 2 where CPU-time consumption decreases for increasing n [12].

IMSL's algorithm consists of four steps (estimation of a condition number has been neglected):

(1) preserving a copy of the input matrix indicated by straight line;

(2) computation of scaling factors represented by the hatched area;

(3) searching for pivot index, updating the pivot column, interchanging rows and rank-one update of the lower right block for the first $n - 1$ columns;

(4) forward/backward substitution marked by two dashed lines at the lower end of the figure.

Irregularities in performance cannot be reported for this routine. For $n = 601$, F04AAF operates at a speed of 95 MFLOPS while the routine L2LRG achieves a rate of 103 MFLOPS.

On IBM, DL2LRG performs best using the analogous sequence of Level 2 BLAS. Again, the storage access pattern of subroutine DL2LRG shows the improved locality. The poor perfor-

Table 4
CPU-time consumption of iterative refinement (real quadratic case)

N	IBM 3081		CRAY X-MP/22	
	F04AEF	DI2ARG	F04AEF	L2ARG
300	3*0.31 sec.	1*1.12 sec.	3*0.24 sec.	1*0.10 sec.
350	3*0.43 sec.	1*1.55 sec.	3*0.33 sec.	1*0.13 sec.
400	3*0.58 sec.	1*2.05 sec.	3*0.43 sec.	1*0.18 sec.

mance of LU, LUS compared with the library routines is mainly caused by cache misses. For $n = 400$, the number of cache misses is about 25 times as large as for DL2LRG.

On CRAY, the costs of one iteration performed by F04AEF are of the same amount as for F04ABF in the symmetric case. In contrast to NAG, IMSL has now changed the implementation of the extended precision dot product (SDSDOT). Here, the summation can be done in one step and SDSDOT proves to be more efficient than X03AAF. For $n \leqslant 100$ (200), one iteration of the IMSL (NAG) routine is more expensive than the solution of a linear system of same order.

The iterative refinement of F04AEF on the IBM is slightly less efficient than for F04ABF. Increased rowwise access to the coefficient matrix leads to a raised number of cache faults. The IMSL approach based on DQDDOT proves to be equivalent to the performance of DQDOTI, DQDOTA in the positive definite case.

For $n = 601$, F04AEF operates a rate of 34 MFLOPS (3 iterations) while the routine L2ARG achieves a speed of 90 MFLOPS (1 iteration).

The relative error in the solution computed by F04AAF and (D)L2LRG is 10^{-12} on IBM and 10^{-10} on CRAY.

5. Solution of complex linear systems

The solution of linear systems with a complex nonsingular matrix can be achieved by:

- (D)L2LCG (IMSL). First, the LU-factorization is computed with scaled partial pivoting (L2CCG). The solution is determined by forward/backward substitution (LFSCG). An estimate for the condition of the problem is provided.
- F04ADF (NAG). The routine decomposes the matrix using Crout's factorization with partial pivoting. The solution is found by forward/backward substitution.
- (D)L2ACG (IMSL). First, the LU-factorization is computed with scaled partial pivoting (L2CCG). The solution is determined by forward/backward substitution (LFSCG). Iterative refinement is used to improve the accurracy of the computed solution (LFICG). An estimate for the condition of the problem is provided.

On CRAY, differences can be observed between L2LCG and F04ADF for large order problems analogous to the real general case; the corresponding Level 2 BLAS is used. Again, iterative refinement becomes the dominating part because complex extended precision is simulated by real extended arithmetic. NAG does not provide a high-accuracy solver.

In contrast to the real general case, no significant differences in performance can be observed on IBM.

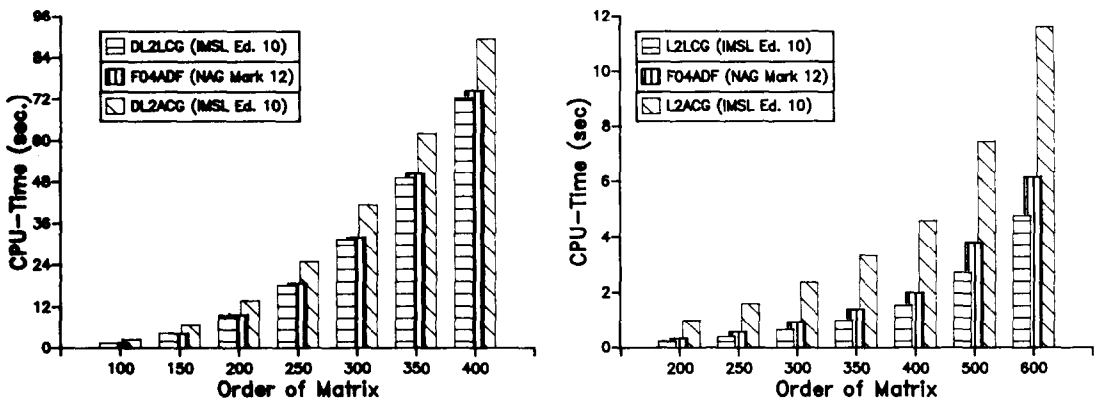

Fig. 4. CPU-time for the solution of linear systems (complex quadratic case) on IBM 3081 and CRAY X-MP/22.

The relative error in the solution computed by F04ADF and (D)L2LCG is 10^{-13} on IBM and 10^{-11} on CRAY (see Fig. 4).

6. Real symmetric eigenvalue problems

In contrast to the previously mentioned problems of matrix multiplication resp. solution of linear systems, eigenvalue problems $Ax = \lambda x$, $x \neq 0$, and $A = A^{\mathrm{T}} \in \mathbb{R}^{n \times n}$, can only be solved numerically by iterative methods [12]. Two variants of the problem are considered here:

(1) computation of the n real eigenvalues $\{\lambda_k\}$;

(2) complete solution of the eigenvalue problem requiring also the computation of an orthonormal set of the n eigenvectors $\{x_k\}$.

A solution of these two problems can be achieved by the subroutines:

- (D)E2LSF, (D)E2CSF (IMSL). The reduction of the coefficient matrix to symmetric tridiagonal form is achieved by orthogonal similarity transformations; the reduced problem is solved by the QL-method (IMTQL2 from EISPACK). The second routine also accumulates eigenvectors and passes them to a separate subroutine yielding the eigenvectors of the original problem by back transformation.
- F02AAF, F02ABF (NAG). The first subroutine represents a recommended variant of the EISPACK path for the computation of the eigenvalues of a full real symmetric matrix; the QL-method (TQL1) is used instead of TQLRAT. F02ABF is an implementation of the recommended EISPACK path (TRED2 and TQL2) calculating eigenvalues and -vectors [18, pp. 212–240].

On both computer systems, NAG performs better than IMSL. On CRAY, differences are caused by the QL-algorithm; IMSL uses a QL-algorithm with implicit shift strategy whereas NAG has chosen TQL1 based on an explicit shift strategy. According to the EISPACK Guide, both approaches have implemented different stopping criteria; for the test problem under consideration ($n = 300$), NAG performs 375 QL-iterations while IMSL requires 406 iterations. If eigenvalues are only required, then the NAG implementation of the iterative part is about twice as fast as the corresponding IMSL routine. The computation of eigenvalues and eigenvectors

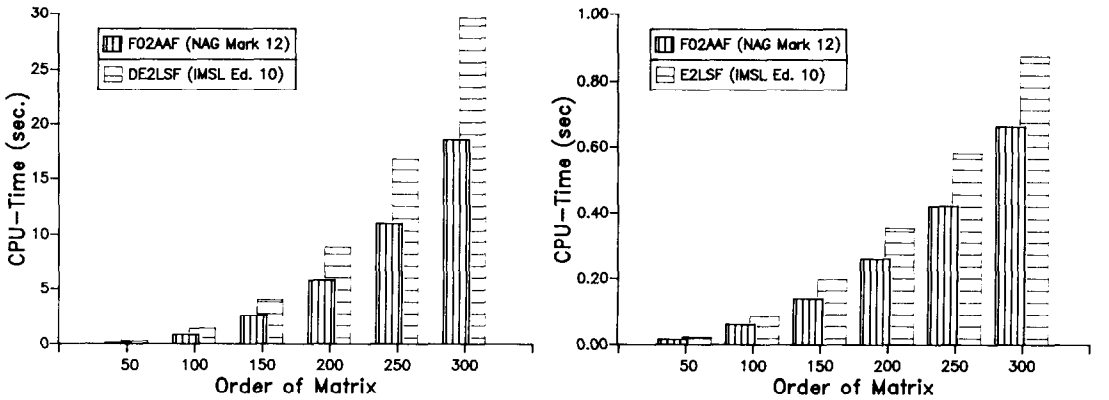

Fig. 5. CPU-time for the computation of the eigenvalues of a symmetric matrix on IBM 3081 and CRAY X-MP/22.

leads to relatively less marked differences in CPU-time consumption because now, the accumulation of similarity transformations represents the dominating subtask; both, IMSL and NAG provide similar implementations for it. But, one should recall that a larger number of QL-iterations leads to an increased execution time for this subtask. The libraries have identical reduction parts; Level 2 BLAS SSYMV (symmetric matrix-vector product) and SSYR2 (symmetric matrix rank-two update) are called. For IMSL, more information on the distribution of CPU-time over subtasks of the symmetric eigenvalue problem is given in Section 7 (Fig. 9).

On IBM, both libraries use the same approach as on CRAY. In contrast to CRAY, the differences are now caused by the reduction parts (TRED1) which always consume a nonnegligible portion of the CPU-time; the analogous sequence of Level 2 BLAS is called. It turns out, that NAG's implementation of DSYMV and DSYR2 is more efficient.

On IBM, the computation of eigenvalues and -vectors costs about four times as much time as the calculation of eigenvalues only. This ratio reduces to a value of three on the CRAY. For $n = 641$, F02AAF operates at a rate of 90 MFLOPS (118 MFLOPS for F02ABF), while the routine E2LSF achieves a speed of 76 MFLOPS (113 MFLOPS for E2CSF) (see Figs. 5 and 6).

7. Real eigenvalue problems

In contrast to the real symmetric eigenvalue problem, a solution of the real eigenvalue problem may lead to pairs of complex conjugate eigenvalues and -vectors. This more general problem can be solved by:

- (D)E2LRG, (D)E2CRG (IMSL) offering the recommended EISPACK paths. The matrix is balanced and then reduced to upper Hessenberg form by orthogonal similarity transformations. The reduced problem is solved by the QR-algorithm. The eigenvectors are normalized.
- F02AFF, F02AGF (NAG) reducing the coefficient matrix to Hessenberg form by stabilized elementary similarity transformations. For the rest, the eigenvalue problem is treated by the same solution path as used above. Both routines dispense with balancing. The eigenvectors are normalized so that the sum of the squares of the moduli of the elements is equal to one and the element of largest modulus is real. This ensures that real eigenvalues have real eigenvectors [18, pp. 9–395].

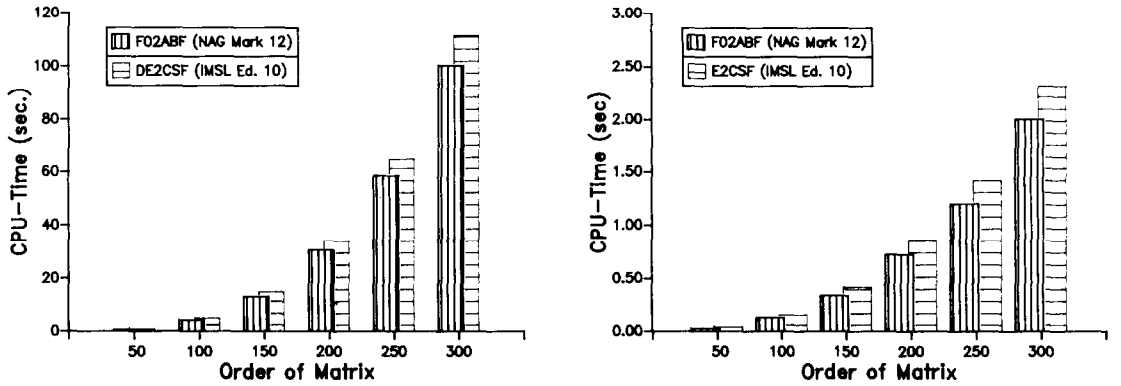

Fig. 6. CPU-time for the computation of the eigenvalues and -vectors of a symmetric matrix on IBM 3081 and CRAY X-MP/22.

Again, differences in CPU-time consumption can be observed on both computer systems. For the IBM version of the NAG library, the computation of eigenvalues and -vectors costs about twice as much time as the calculation of eigenvalues only. This ratio reduces to a value of 1.6 on the CRAY. For $n = 545$, F02AFF operates at a rate of 104 MFLOPS (114 MFLOPS for F02AGF), while the routine E2LRG achieves a speed of 64 MFLOPS (72 MFLOPS for E2CRG).

In the IMSL library, the reduction to upper Hessenberg matrix is based on Level 2 BLAS SGEMV (DGEMV) and SGER (DGER) performing the orthogonal similarity transformations. The NAG library uses a different approach; here, the reduction is carried out by stabilized elementary similarity transformations using SGEMV (DGEMV) and STRSV (DTRSV). The EISPACK Guide states that a replacement of elementary stabilized similarities by orthogonal similarities may increase execution time by about 20% [17]. For the library software under consideration, this difference has been observed to be much larger.

On IBM, differences in performance are mainly caused by the reduction part. For $n = 300$, the IMSL subroutine consumes more than twice the CPU-time of the NAG routine. On CRAY, the situation is reversed; the observed differences originate from the QR-algorithms. For $n = 300$,

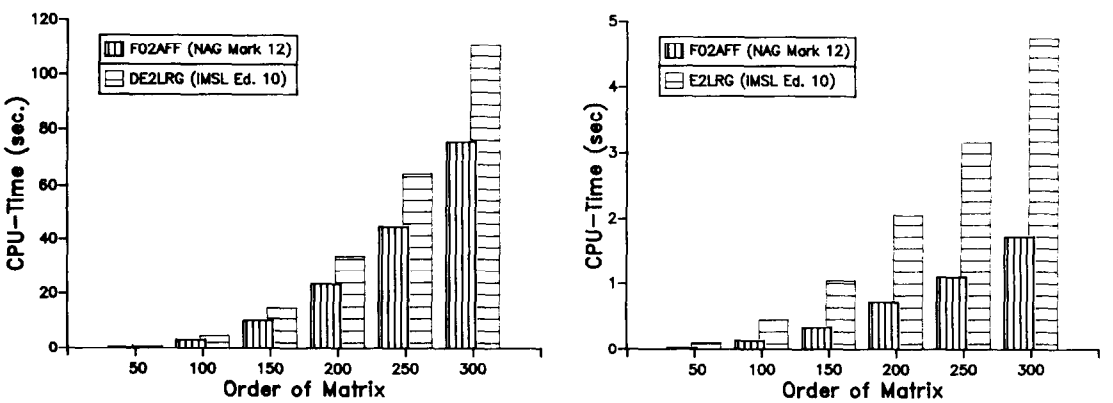

Fig. 7. CPU-time for the computation of the eigenvalues of a real matrix on IBM 3081 and CRAY X-MP/22.

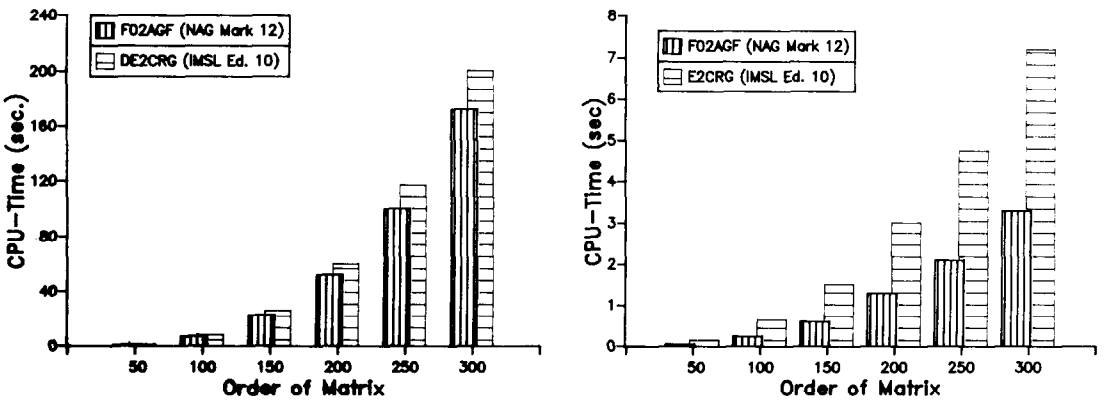

Fig. 8. CPU-time for the computation of the eigenvalues and -vectors of a real matrix on IBM 3081 and CRAY X-MP/22.

the execution times of the corresponding subroutines differ by more than a factor of two which is due to poor code generated by the compiler CFT77 version 1.3. A recompilation of the IMSL subroutine E3CRH with CFT 1.15 leads to a performance similar to the one achieved by NAG. Compiler artifacts of this type can only be detected and overcome if the source code has been supplied to the computer center (see Figs. 7 and 8).

For the CRAY version of IMSL, the distribution of CPU-time over subtasks of the real eigenvalue problem is given by Fig. 9 and may be compared with the results for the real symmetric case. Two principal subtasks have to be considered: the reduction of the coefficient matrix to Hessenberg form or symmetric tridiagonal form, respectively, and the QR-algorithm for the reduced matrix. Balancing represents a less time consuming part and is negligible within

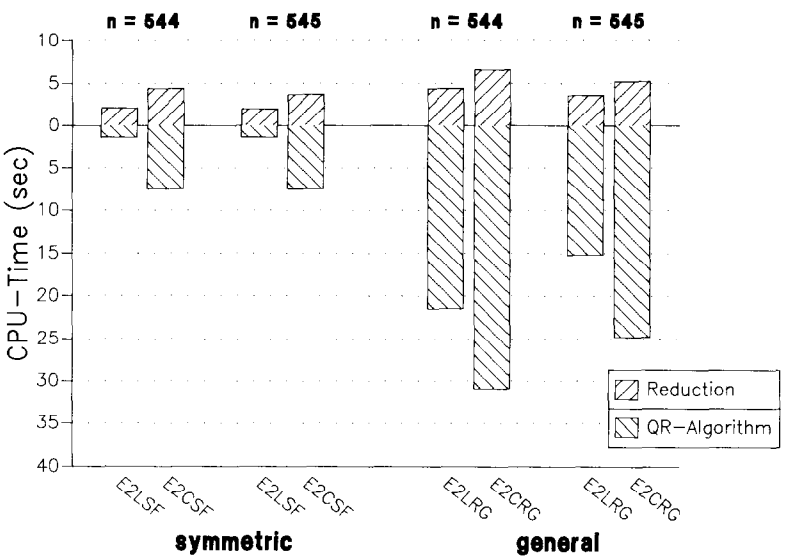

Fig. 9. CPU-time distribution on subtasks for real eigenvalue problems on CRAY X-MP/22.

this context. The CPU-time consumption of the reduction part is plotted in the upper half of Fig. 9; the corresponding time for the QR-algorithm is plotted below the reference line. Problems of size $n = 544$ and $n = 545$ are analyzed. Times for these orders are not affected by a strongly differing number of QR-steps. For the solution of real general eigenvalue problems on a CRAY, one can observe that the CPU-time consumption of the subroutines E2LRG and E2CRG is sensitive to bank conflicts which are indicated by decreasing times for increasing n. This effect has also been observed for the real symmetric eigenvalue problem; but, it is more marked for the real general case. For IMSL, an increased number of bank conflicts originate from the reduction of the input matrix to Hessenberg form; here, the premultiplication part of the similarity transformation is sensitive to bank conflicts caused by rowwise access to the coefficient matrix. The NAG approach always leads to columnwise access of the matrix avoiding bank conflicts. Both, the IMSL and NAG implementations of the QR-algorithm are sensitive to bank conflicts. Supplementary information on the corresponding NAG software is given in reference [10].

8. Conclusions

For a small subset of linear algebra problems, the performance (accuracy and runtimes) of library subroutines (IMSL, NAG) on a CRAY X-MP/22 and IBM 3081 has been examined; special attention has been paid to large-order test matrices. Publications suitable to expand the perspective have been taken into account.

The report shows that, for the test problems under consideration, the library routines do not differ significantly in the accuracy of the computed results. But differences can be observed for the CPU-time consumption to solve a particular problem. Both libraries benefit from Level 2 BLAS and on vector computers, they have at least partially implemented the technique of unrolling. SGEMV of IMSL has unrolled the vector loop to a depth of 16 while NAG has implemented unrolling to a depth of four. For STRSV, the situation is reversed; again, NAG uses unrolling to a depth of four but IMSL dispenses with this technique. Unrolling could also improve the performance of library software on a scalar computer. IMSL has partially implemented this approach at Level 1 BLAS (DDOT depth five, DAXPY depth four) [6]. Thus, merging for each computer architecture the most efficient BLAS operations together should improve the performance of library software calling these routines.

The difference between a brute-force approach represented by the *ijk*-version of matrix multiplication and a sophisticated highly optimized routine from a library is illustrated by the corresponding megaflop rates; for $n = 600$, MRRRR achieves 170 MFLOPS, while the standard *ijk*-implementation results in 55 MFLOPS. This simple example already demonstrates the potential a library may offer to its users.

Weak parts of a library can be detected by a comparison of performance results for carefully selected test problems. Additional results giving advice on the selection of subroutines from the various libraries can be reported for the field of fast Fourier transforms and special functions. However, the tests should be extended to other library chapters in order to deal with all aspects of library software.

Acknowledgements

The authors are grateful to Monika Strompen for running the jobs on the MVS systems. Siegfried Knecht helped substantially to produce the storage access patterns of the subroutines LU, LUS and L2LRG. The authors also would like to thank the referee for his very helpful comments.

References

[1] Th. J. Aird, The IMSL Library, in: W.R. Cowell, Ed., *Sources and Development of Mathematical Software* (Prentice-Hall, Englewood Cliffs, NJ, 1984) 264–301.
[2] CRAY X-MP and CRAY-1 Computer Systems, Programmer's Library Reference Manual SR-0113, 1986.
[3] J.J. Dongarra, J. DuCroz, S. Hammarling and R.J. Hanson, An extended set of Fortran basic linear algebra subprograms, *ACM Trans. Math. Software* **14** (1988) 1–18.
[4] J.J. Dongarra and S.C. Eisenstat, Squeezing the most out of an algorithm in CRAY Fortran, *ACM Trans. Math. Software* **10** (1984) 219–230.
[5] J.J. Dongarra, F.G. Gustavson and A. Karp, Implementing linear algebra algorithms for dense matrices on a vector pipeline machine, *SIAM Rev.* **26** (1984) 91–112.
[6] J.J. Dongarra and A.R. Hinds, Unrolling Loops in FORTRAN, *Software — Practice and Experience* **9** (1979) 219–226.
[7] J.J. Dongarra, C.B. Moler, J.R. Bunch and G.W. Stewart, *LINPACK Users' Guide* (SIAM, Philadelphia, 1979).
[8] B. Ford and J.C.T. Pool, The evolving NAG library service, in: W.R. Cowell, Ed., *Sources and Development of Mathematical Software* (Prentice-Hall, Englewood Cliffs, NJ, 1984) 375–395.
[9] R.T. Gregory and D.L. Karney, *A Collection of Matrices for Testing Computational Algorithms* (Wiley-Interscience, New York, 1969).
[10] J.-Fr. Hake and W. Homberg, Solution of real eigenvalue problems on a CRAY vector computer, 1988 Spring Proc. CRAY User Group Meeting (1988) 344–350.
[11] J.-Fr. Hake and W. Homberg, ZAMLIB Documentation, KFA/ZAM Jülich, 1988.
[12] J.-Fr. Hake and W. Homberg, Linear algebra software on a vector computer, *Parallel Comput.* **10** (1) (1989) 65–81.
[13] IMSL MATH/LIBRARY User's Manual, IMSL, Houston, 1987.
[14] C.L. Lawson, R.J. Hanson, D.R. Kincaid and F.T. Krogh, Basic linear algebra subprograms for Fortran usage, *ACM Trans. Math. Software* **5** (1979) 308–323.
[15] R.E. Lord, J.S. Kowalik and S.P. Kumar, Solving linear algebraic equations on an MIMD computer, *J. Assoc. Comput. Mach.* **30** (1983) 103–117.
[16] NAG Library Manual Mark 12, NAG, Oxford, 1987.
[17] B.T. Smith et al., *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science **6** (Springer, Berlin, 1976).
[18] J.H. Wilkinson and C. Reinsch, *Handbook for Automatic Computation, Volume II, Linear Algebra* (Springer, Berlin, 1971).